Level Up! CI/CD

Continuous Integration and Deployment

Curtis Spendlove

Contents

Preface v

1 Introduction 1
1.1 Fork / Cone Project . . . 1

2 A Basic Test Harness with GitLab 3
2.1 Sample Project . . . 4
2.2 Add .gitlab-ci.yml . . . 7
2.2.1 Commit, Push, and Merge . . . 8
2.3 Add Linting . . . 9
2.3.1 Auto-Linting . . . 13
2.3.2 Fail Softly . . . 14
2.4 Add Tests . . . 15
2.4.1 Create a Test . . . 17

3 Expanding the GitLab Test Harness 21
3.1 Fix the Linting . . . 21
3.2 Add Code Coverage . . . 24
3.2.1 GitLab Code Coverage . . . 24
3.2.2 Do-It Yourself . . . 25
3.3 Add Code Coverage HTML Report . . . 27
3.4 Add GitLab Pages Job . . . 29
3.4.1 Uncomment the Master Constraint . . . 30
3.5 Tests, Tests, Tests . . . 31
3.6 Badges, Badges! We don't NEED no Stinkin' Badges . . . 37

4 Adding Continuous Deployment to the Pipeline 39

5 BitBucket CI/CD 41

6 GitHub CI/CD 43

Preface

This is an example of “frontmatter”, which comes before the main text of the book.

Chapter 1

Introduction

After the book is completed (or mostly completed) there will be a public repo representing the code in the book. It isnt here, because Im still thinking through the exact form I want it to take (I want it to be as instructional as possible, and that is going against my nature to heavily rebase my code for clean histories.)

:)

1.1 Fork / Cone Project

fork / clone sample project…

push it to github on your account

continue on, brave warrior

IF, you dont want to use the repo, you can use a new express project. Actually I might want to make it so you do this anyway…at least in the GitLab chapters. A simple project to get to know CI; then we can push a much better codebase into CI.

Chapter 2

A Basic Test Harness with GitLab

This is a basic test harness using GitLab as an example.

Note: Im using GitLab for the first portion. This is an introductory to CI/CD for beginner/intermediate programmers (whove finished an excellent set of tutorials or otherwise have some basic experience with coding). Its also good for anyone relatively new to CI/CD workflows. For anyone experienced, you may want to skip to (chapter whatever) for a more piecemeal configuration. (There are some limitations to using a single, overall offering that can be solved by using a more complex CI/CD workflow.)

This book, for now, does lean heavily toward Node.js / Express (in particular the MEAN stack). I plan on either adding appendices (or just writing separate flavors of the book) for other popular frameworks. (Im leaning toward appendices if I can make them make sense…hopefully they make sense as a diff of the modifications that would be required to work for a Rails or Django project, etc.)

(Move intro text blocks into a what to expect or how to navigate the book section of the introduction.)

The next few chapters will serve as an introduction to Continuous Integration, Delivery, and Deployment, using GitLab and Heroku as our vehicles. If you use a public project and dont do anything excessive on Heroku, following these chapters should be free.

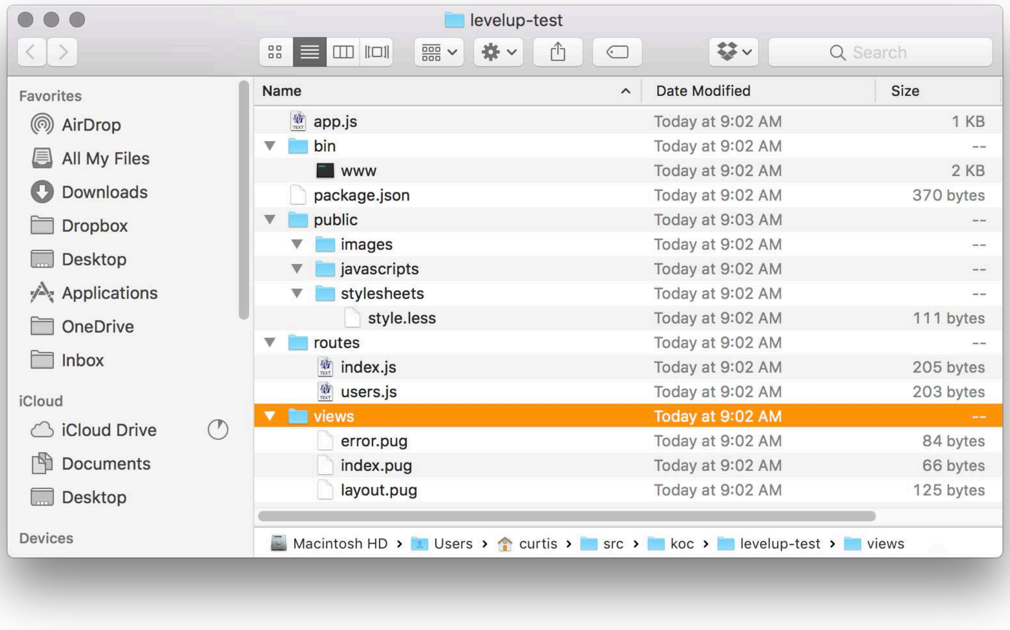
For the purposes of the GitLab chapters, well be creating a very basic Node.js / Express app. If youre unfamiliar with Node.js please feel free to either download the base repository (without CI, ready for use throughout this guide) or use your own repo.

(Note: if your code is in a language directly compatible with `npm`, you should be fine with the examples as-is. If youre using a different language like Ruby, Python, etc; youll need to tweak the build scripts for your build / test tools. Future versions of this book will help identify and resolve these issues for multiple languages.)

(Move the previous disclaimer to the front, as well.)

2.1 Sample Project

Well be creating a basic web application project using Node.js / Express. We will then place it under version control with GitLab and add in CI/CD phases. This will demonstrate a reasonable number of the sorts of issues youll encounter adding (or expanding) CI/CD to any project.



Name	Date Modified	Size
app.js	Today at 9:02 AM	1 KB
▼ bin	Today at 9:02 AM	--
www	Today at 9:02 AM	2 KB
package.json	Today at 9:02 AM	370 bytes
▼ public	Today at 9:03 AM	--
▼ images	Today at 9:02 AM	--
▼ javascripts	Today at 9:02 AM	--
▼ stylesheets	Today at 9:02 AM	--
style.less	Today at 9:02 AM	111 bytes
▼ routes	Today at 9:02 AM	--
index.js	Today at 9:02 AM	205 bytes
users.js	Today at 9:02 AM	203 bytes
▼ views	Today at 9:02 AM	--
error.pug	Today at 9:02 AM	84 bytes
index.pug	Today at 9:02 AM	66 bytes
layout.pug	Today at 9:02 AM	125 bytes

Youll need `node`, `npm`, and `express` installed on your development machine in order to create the base project and to locally run portions of the build chain (localized linting, testing, etc). They can be easily installed on most operating systems, and if youd rather not install them on your local machine, a VM or Cloud IDE are excellent options.

Start with an empty subdirectory in whichever directory houses your development projects:

Listing 2.1: Create Sample Project

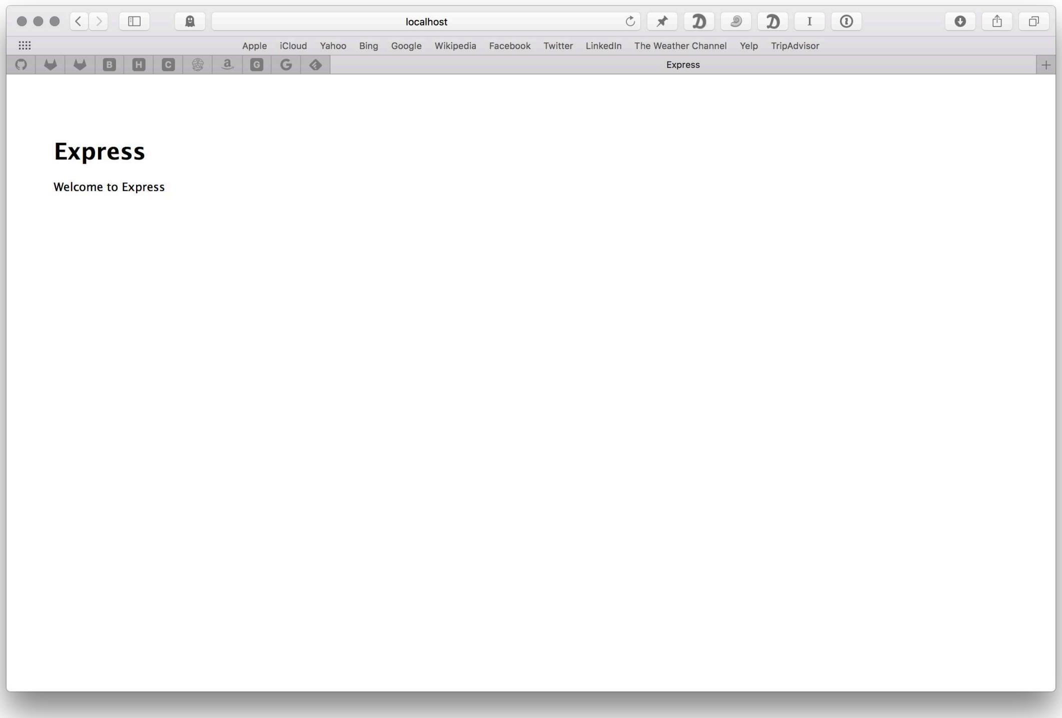
```
$ mkdir levelup-test
$ cd levelup-test
$ express --css=less --view=pug --git
```

I always smoke test a project the first time it gets on my hard drive, regardless of whether I create it myself (like we just did) or clone it from a remote. This project doesnt even have a test suite yet, so our smoke test is manual.

Listing 2.2: Manual Smoke Test

```
$ npm install

npm notice created a lockfile as package-lock.json. You should commit this file.
added 173 packages in 9.374s

$ npm start

> levelup-test@0.0.0 start /Users/curtis/src/koc/levelup-test
> node ./bin/www
```

We can then visit our application on the default port of 3000.



Express

Welcome to Express

Place it under version control, create a GitLab remote, and push the code.

Listing 2.3: Version and Push to GitLab

```
$ git init
$ git remote add origin git@gitlab.com:[group]/[project-name].git
$ git add --all
$ git commit [...]
$ git push -u origin master
```

2.2 Add .gitlab-ci.yml

When getting to know a feature, its often a good idea to start with any templates provided by the tool makers. GitLab has a button in the project overview to walk through this (they also publish the templates in a repo).

I expect that tool changes over time as best practices are discovered and refined. As of this books publishing, the below script is the resulting template:

Listing 2.4: .gitlab-ci.yml

```
# This file is a template, and might need editing before it works on your project.
# Official framework image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/node/tags/
image: node:latest

# Pick zero or more services to be used on all builds.
# Only needed when using a docker container to run your tests in.
# Check out: http://docs.gitlab.com/ce/ci/docker/using_docker_images.html#what-is-a-service
services:
  - mysql:latest
  - redis:latest
  - postgres:latest

# This folder is cached between builds
# http://docs.gitlab.com/ce/ci/yaml/README.html#cache
cache:
  paths:
  - node_modules/

test_async:
  script:
   - npm install
   - node ./specs/start.js ./specs/async.spec.js

test_db:
  script:
   - npm install
   - node ./specs/start.js ./specs/db-postgres.spec.js
```

This initial template is good to start, but we will modify it over time to enforce our preferences. The first minor changes involve the services and jobs:

- delete `services` section (we wont be linking services in this sample)
- change `test_async` to `test` (I like concise names)
- delete `node` call from `test` job (we dont have tests yet)
- delete `test_db` (we wont be testing the db in this sample)

Listing 2.5: .gitlab-ci.yml

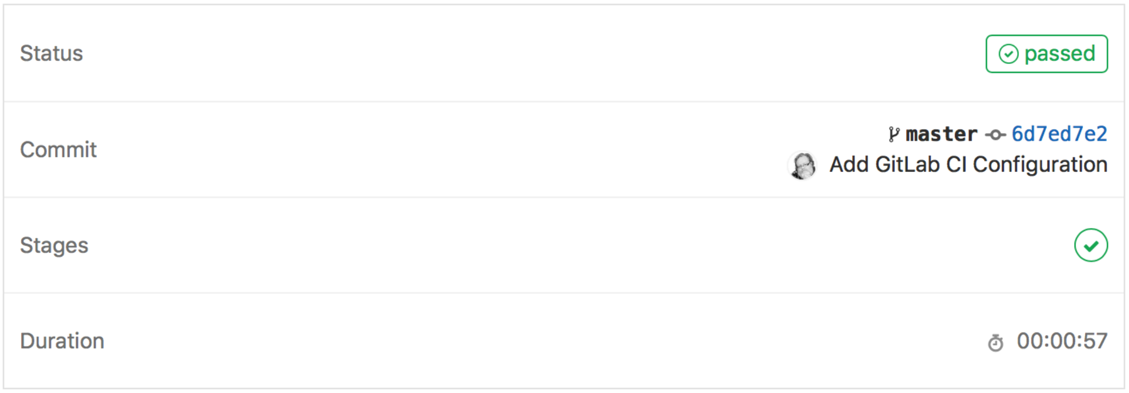
```
# This file is a template, and might need editing before it works on your project.
# Official framework image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/node/tags/
image: node:latest

# This folder is cached between builds
# http://docs.gitlab.com/ce/ci/yaml/README.html#cache
cache:
  paths:
  - node_modules/

test:
  script:
   - npm install
```

(Note: This is a book about CI/CD, not about testing. The basic framework we setup here will work with expanded configurations to flawlessly test very complex setups with multiple databases and other attached services. Start out small, grow big. In fact, I recommend if youre going to be adding CI/CD to a large flagship project, you should instead start out small on a test project.)

2.2.1 Commit, Push, and Merge

The GitLab interface should now show the branch running the pipeline. It should be triggered whenever you create an MR, push a code update to the branch, or merge back into the parent branch.

This allows us to enforce constraints to virtually guarantee quality code before its merged back into the parent branch and, up the chain, deployed into production.



Status	✓ passed
Commit	master 6d7ed7e2 Add GitLab CI Configuration
Stages	✓
Duration	00:00:57

It doesnt do much yet, but it *does* trigger, and thats a victory. Grab your favorite beverage and pat yourself on the back. . . the easy part is done. ;)

2.3 Add Linting

One of the most wonderful options a CI pipeline provides is an easy way to lint code.

Linting: . . . (define)

Like most auxiliary processes, linting is performed via an external tool. There are many, and they vary per language. My preference for JavaScript is `eslint`.

Listing 2.6: Install eslint

```
$ npm install --save-dev eslint

+ eslint@4.7.0
added 127 packages in 80.414s
```

Adding a package to your project with `npm install` adds its binaries to the `.node_modules` subdirectory. This allows you to call it manually, or through scripts.

Listing 2.7: Run eslint

```
$ ./node_modules/.bin/eslint .

Oops! Something went wrong! :(

ESLint: 4.6.1.
ESLint couldn't find a configuration file. To set up a configuration file for this
project, please run:

    eslint --init

ESLint looked for configuration files in /Users/curtis/src/koc/levelup-test and its
ancestors.

If you think you already have a configuration file or if you need more help, please s
top by the ESLint chat room: https://gitter.im/eslint/eslint
```

Well, at least it starts!

You can, of course, use a preferred template for your configuration file if you have one. If not, fire up `eslint -init`.

Listing 2.8: Initialize eslint

```
$ eslint --init
? How would you like to configure ESLint? Answer questions about your style
? Are you using ECMAScript 6 features? Yes
? Are you using ES6 modules? Yes
? Where will your code run? Browser
? Do you use CommonJS? No
? Do you use JSX? No
? What style of indentation do you use? Spaces
? What quotes do you use for strings? Single
? What line endings do you use? Unix
? Do you require semicolons? Yes
? What format do you want your config file to be in? JSON
Successfully created .eslintrc.json file in /Users/curtis/src/koc/levelup-test
```

You can see here that I went with the standard template (which I usually customize slightly) and I prefer JSON format.

The tool finishes up by installing any needed plugins and creates an `.eslintrc.json` file.

Listing 2.9: Resulting .eslintrc.json

```
{
    "env": {
        "browser": true,
        "es6": true
    },
    "extends": "eslint:recommended",
    "parserOptions": {
        "sourceType": "module"
    },
    "rules": {
        "indent": [
            "error",
            4
        ],
        "linebreak-style": [
            "error",
            "unix"
        ],
        "quotes": [
            "error",
            "single"
        ],
        "semi": [
            "error",
            "always"
        ]
    }
}
```

This configuration would be a pretty good start, but I recommend a slightly simplified one while learning.

Listing 2.10: Suggested .eslintrc.json

```
{
  "rules": {},
  "env": {
    "es6": true,
    "browser": true
  },
  "extends": "eslint:recommended"
}
```

(You're welcome to use your preference, if you have one. However, if you're following the tutorial exactly, it's a good idea to stick to the suggested config.)

As usual, well be customizing this over time, but its a pretty good start. We can now rerun `./node_modules/.bin/eslint .`

And get a bunch of errors (mine shows 21).

Listing 2.11: Suggested .eslintrc.json

```
$ ./node_modules/.bin/eslint .

/Users/curtis/src/koc/levelup-test2/app.js
   1:15  error  'require' is not defined                      no-undef
   2:12  error  'require' is not defined                      no-undef
   3:5   error  'favicon' is assigned a value but never used  no-unused-vars
   3:15  error  'require' is not defined                      no-undef
   4:14  error  'require' is not defined                      no-undef
   5:20  error  'require' is not defined                      no-undef
   6:18  error  'require' is not defined                      no-undef
   7:22  error  'require' is not defined                      no-undef
   9:13  error  'require' is not defined                      no-undef
  10:13  error  'require' is not defined                      no-undef
  15:28  error  '__dirname' is not defined                    no-undef
  24:34  error  '__dirname' is not defined                    no-undef
  25:34  error  '__dirname' is not defined                    no-undef
  38:33  error  'next' is defined but never used              no-unused-vars
  48:1   error  'module' is not defined                       no-undef

/Users/curtis/src/koc/levelup-test2/routes/index.js
  1:15  error  'require' is not defined          no-undef
  5:36  error  'next' is defined but never used  no-unused-vars
  9:1   error  'module' is not defined           no-undef

/Users/curtis/src/koc/levelup-test2/routes/users.js
  1:15  error  'require' is not defined          no-undef
  5:36  error  'next' is defined but never used  no-unused-vars
  9:1   error  'module' is not defined           no-undef

 21 problems (21 errors, 0 warnings)
```

The *good* news is that the above items are pretty easy to fix. The *better* news is that we can allow this to fail for now.

2.3.1 Auto-Linting

Its pretty painful to remember to type `./node_modules/.bin/eslint .` every time we want to lint. It also violates best practices for CI/CD. If its not *automatic*, its not *continuous*.

Fortunately, `npm` makes this pretty easy. We can add any number of aliases with the scripts key in `package.json`:

Listing 2.12: Add "lint" Script to package.json

```
  "scripts": {
    "start": "node ./bin/www",
    "lint": "./node_modules/.bin/eslint ."
  }
```

Now a simple `npm run lint` provides you the extensive list of linting violations. Yay, tools!

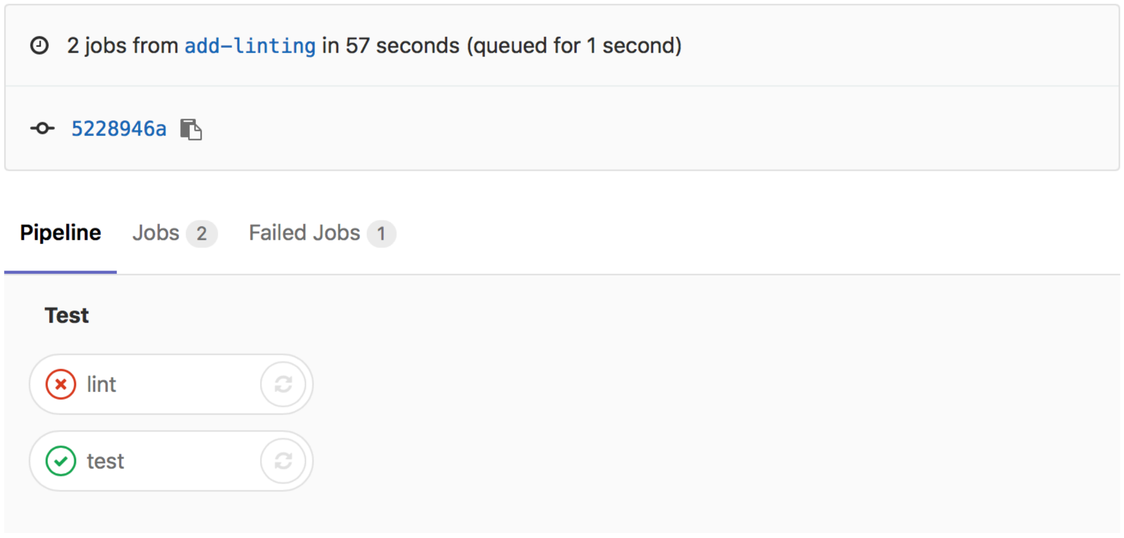
More importantly, it also gives us an easy command to add to a linting job in the integration pipeline.

Listing 2.13: Add "lint" Task to .gitlab-ci.yml

```
lint:
  script:
   - npm install
   - npm run lint
```

Commit, Push, Fail

This, of course, results in a failure.


2 jobs from add-linting in 57 seconds (queued for 1 second)

5228946a

If you click and drill down into the results of the lint task, Im sure you can verify your guess as to why. If it fails locally, rest assure it will fail in the integration pipeline.

Ill provide the code to fix this later (or you can grab it from the online repo if youre impatient). But for now, lets introduce `allow_failure`.

2.3.2 Fail Softly

We are currently failing hard, which triggers an error condition in the pipeline, stopping anything further in the chain from executing (the rest of the parallel tasks in the same phase of the pipeline will finish and indicate their error status—notice test passed).

Fail fast is a basic best practice in software development. Its functioning as we want it to. However, linting is for humans, not computers. The bots dont care if our code is pretty, they care if it runs.

(Personally, I feel that in most cases one should strive to write exemplary code. Therefore I prefer to require linting in all codebases where its possible. However, teams must make this decision on what makes sense for them. And its often a process over time, especially in legacy codebases.)

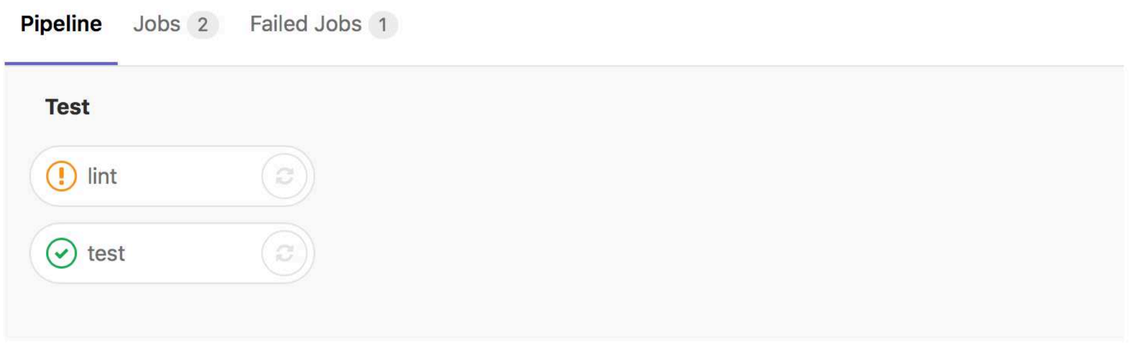
Fortunately, failing with a warning is a simple matter in most tools. In GitLab we add `allow_failure: true`:

Listing 2.14: Allow Failure: True

```
lint:
  script:
   - npm install
   - npm run lint
  allow_failure: true
```

Commit, Push, Succeed (Kinda)

Your pipeline is still angry with you, but its didnt take the trash out angry, instead of forgot your anniversary again angry.



Pull / Merge Request

Were done with this section, merge that sucker into its parent. That yellow warning **will** bother you, and it should. This isnt a permanent solution. Its a stop-gap.

2.4 Add Tests

A test harness isnt very useful without tests.

Before we conclude this chapter, we need to perform a similar task for the testing tools as we did for the linting ones. Well use a pretty common suite of testing tools for Node / Express: `mocha`, `chai`, and `superagent`. Well augment them as we go along with extra tools to support more best practices.

We follow the same pattern as earlier. Install, familiarize, add to `npm`, add to pipeline.

Listing 2.15: Install Testing Tools

```
$ npm install --save-dev mocha chai superagent
+ superagent@3.6.0
+ chai@4.1.2
+ mocha@3.5.3
added 31 packages in 40.155s
```

I prefer to keep my tests in a `test` directory when dealing with Node projects. We dont currently have that, so well get warnings (but well fix that shortly).

Listing 2.16: Manual Test Check

```
$ ./node_modules/.bin/mocha ./test
```

As always, we can add this into our package.json.

Listing 2.17: Add Test Script to package.json

```
  "scripts": {
    "start": "node ./bin/www",
    "lint": "./node_modules/.bin/eslint .",
    "test": "mocha ./test"
  }
```

And check it.

Listing 2.18: Check npm Test Script

```
$ npm run test
```

(npm gives us a shortcut for the test script)

Listing 2.19: npm test alias

```
$ npm test
```

And we can add the test job to the .gitlab-ci.yml file:

Listing 2.20: Add Test Job to .gitlab-ci.yml

```
test:
  script:
   - npm install
   - npm test
```

After this, the suite should try to run the test (push and drill down to check). It should fail, as no tests were executed. (Though various systems may treat no tests as a pass.)

2.4.1 Create a Test

Well create a test (which will fail linting miserably, but thats acceptable for now—well get back to linting).

Listing 2.21: Create test/index.test.js

```
$ mkdir test
$ touch test/index.test.js
```

Listing 2.22: Sample test/index.test.js

```
var assert = require('assert');

describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present', function() {
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

This test is straight from the Mocha homepage, and it should pass with flying colors (mostly green colors):

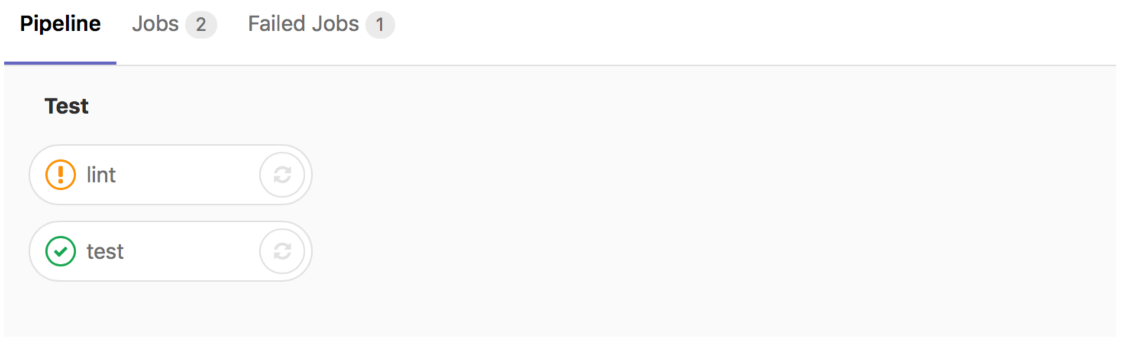
Listing 2.23: npm test passing

```
$ npm test

> levelup-test@0.0.0 test /Users/curtis/src/koc/levelup-test
> mocha ./test



  Array
    #indexOf()
        should return -1 when the value is not present


  1 passing (7ms)
```

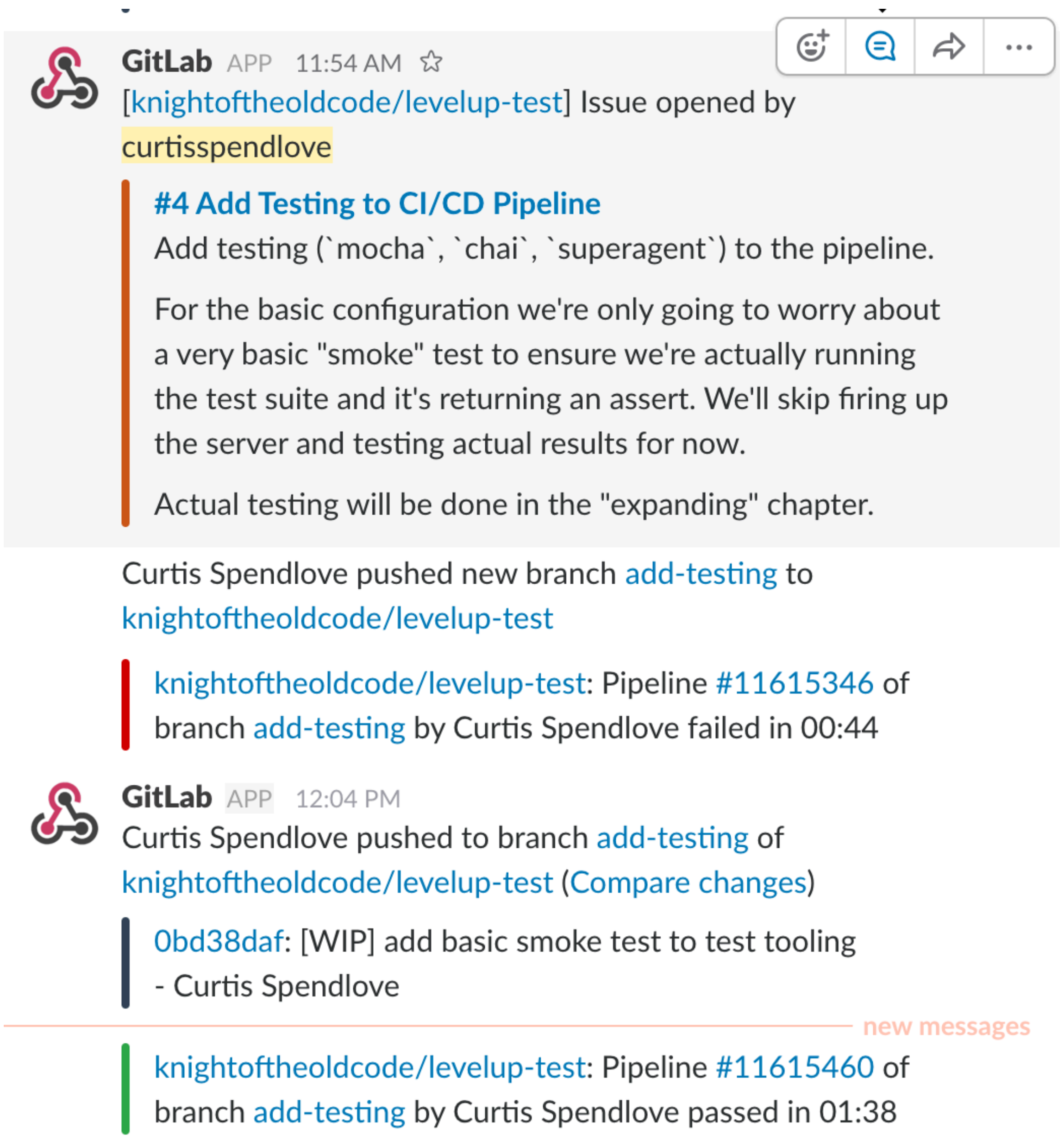
Itll do, Donkey, itll do.

Commit, Push, Succeed

Success!



You can drill down if you wish. It is good to verify the server is performing what you expect it to. (I knew it succeeded even before drilling down or typing the above paragraph, slack told me so. I highly recommend getting up a clearinghouse for notifications.)


GitLab APP 11:54 AM
[knightoftheoldcode/levelup-test] Issue opened by curtisspendlove

#4 Add Testing to CI/CD Pipeline
Add testing (`mocha`, `chai`, `superagent`) to the pipeline.

For the basic configuration we're only going to worry about a very basic "smoke" test to ensure we're actually running the test suite and it's returning an assert. We'll skip firing up the server and testing actual results for now.

Actual testing will be done in the "expanding" chapter.

Curtis Spendlove pushed new branch add-testing to knightoftheoldcode/levelup-test

knightoftheoldcode/levelup-test: Pipeline #11615346 of branch add-testing by Curtis Spendlove failed in 00:44

GitLab APP 12:04 PM
Curtis Spendlove pushed to branch add-testing of knightoftheoldcode/levelup-test (Compare changes)

0bd38daf: [WIP] add basic smoke test to test tooling - Curtis Spendlove

knightoftheoldcode/levelup-test: Pipeline #11615460 of branch add-testing by Curtis Spendlove passed in 01:38

Merge

Were done with this section, and also this chapter. We have a very basic test harness setup. No, it doesnt really *do* much, but its easy to expand. Take a few, relax. Play some Factorio (Im sure your boss would like to be introduced to it.)

In the next chapter, well expand the test harness into something more useful. Well fix the linting, add real tests, and add code coverage (including pushing static results to GL pages).

Chapter 3

Expanding the GitLab Test Harness

In this chapter, we expand the test harness to be more useful, and more accurate.

3.1 Fix the Linting

Im tired of the yellow warning. Im guessing you are too. :) Fortunately, this is a tiny sample project, so its quite easy to take advantage of `eslint` and clean up our code to conform to modern JavaScript styling (ES6, in this case).

Code style is a separate book in and of itself, so as it stands Im just going to provide linted code which should pass our pipeline. I arrived at these modifications through a combination of `eslint`s auto fix features and a few manual styling modifications.

Listing 3.1: app.js

```
var express = require('express');
var path = require('path');
// var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var lessMiddleware = require('less-middleware');
var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(lessMiddleware(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', index);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Listing 3.2: routes/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
```